1 computer systems

This book is about writing well-designed software. To understand software, we must first understand its role in a computer system. Hardware and software work together in a computer system to accomplish complex tasks. Furthermore, computer networks have changed how computers are used, and they now play a key role in even basic software development. This chapter explores a broad range of computing issues, laying the foundation for the study of software development.

chapter objectives

- Describe the relationship between hardware and software.
- Define various types of software and how they are used.
- Identify basic computer hardware and explain what it does.
- Explain how the hardware components execute programs and manage data.
- Describe how computers are connected together into networks to share information.
- Explain the importance of the Internet and the World Wide Web.
- Introduce the Java programming language.
- Describe the steps involved in program compilation and execution.
- Introduce graphics and their representations.

1.0 introduction

We begin our exploration of computer systems with an overview of computer processing, defining some basic terminology and showing how the key pieces of a computer system work together.

basic computer processing

A computer system is made up of hardware and software. The *hardware* components of a computer system are the physical pieces. They include chips, boxes, wires, keyboards, speakers, disks, cables, plugs, printers, mice, monitors, and so on. If you can physically touch it and it can be considered part of a computer system, then it is computer hardware.

key concept: A computer system consists of hardware and software that work together to help us solve problems.

The hardware components of a computer are useless without instructions to tell them what to do. A *program* is a series of instructions that the hardware executes one after another. *Software* includes programs and the data those programs use. Together hardware and software form a tool that we can use to solve problems.

The key hardware components in a computer system are:

- central processing unit (CPU)
- input/output (I/O) devices
- main memory
- secondary memory devices

Each of these hardware components is described in detail in the next section. For now, let's simply examine their basic roles. The *central processing unit* (CPU) is the device that executes the individual commands of a program. *Input/output* (I/O) *devices,* such as the keyboard, mouse, and monitor, allow a person to interact with the computer.

Programs and data are held in storage devices called memory, which fall into two categories: main memory and secondary memory. *Main memory* holds the software while it is being processed by the CPU. *Secondary memory* stores software more or less forever—until it is deliberately erased. The most important secondary memory device of a typical computer system is the hard disk, which is inside the main computer box. A floppy disk is like a hard disk, but it cannot store nearly as much information as a hard disk. Floppy disks are portable. That is, they can be removed or moved from computer to computer as needed. Other portable secondary memory devices include zip disks and compact discs (CDs).

Figure 1.1 shows how information moves among the basic hardware parts of a computer. Suppose you have a program you wish to run. The program is stored on some secondary memory device, such as a hard disk. When you tell the computer to execute your program, a copy of the program is brought in from secondary memory and stored in main memory. The CPU reads the program instructions from main memory. The CPU then executes the instructions one at a time until the program ends. The data that the instructions use, such as two numbers that will be added together, are also stored in main memory. They are either brought in from secondary memory or read from an input device such as the keyboard. During execution, the program may display information to an output device such as a monitor.

key concept
To execute a program, the computer first copies the program from secondary memory to main memory. The CPU then reads the program instructions from main memory, executing them one at a time until the program ends.

The process of executing a program is basic to the operation of a computer. All computer systems work in about the same way.

software categories

There are many types of software. At this point we will simply look at system programs and application programs.

The *operating system* is the main software of a computer. It does two things. First, it provides a *user interface* that allows the user to interact with the machine: to click on an icon, for example, or delete a file. Second, the operating system manages computer resources such as the CPU and main memory. It decides when programs can run, where they are loaded into memory, and how hardware devices communicate. It is the operating system's job to make the computer easy to use and to keep it running well.

key concept
The operating system provides a user interface and manages computer resources.

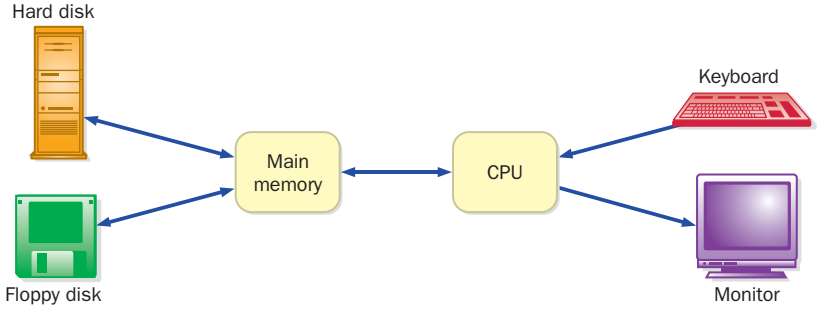


figure 1.1 A simplified view of a computer system

Several popular operating systems are in use today. Windows 98, Windows NT, Windows 2000, and Windows XP are versions of the operating system developed by Microsoft for personal computers. Versions of the Unix operating system are also quite popular, such as Linux. Mac OS is the operating system used on Apple computers.

An *application* is just about any software other than the operating system. Word processors, missile control systems, database managers, Web browsers, and games are all application programs. Each application program has its own user interface that allows the user to interact with that particular program.

The user interface for most modern operating systems and applications is a *graphical user interface* (GUI), which uses graphical screen elements. These elements include:

- *windows,* which are used to separate the screen into distinct work areas
- *icons,* which are small images that represent computer resources, such as a file
- *pull-down menus,* which give the user a list of options
- *scroll bars,* which let the user move up and down in a window
- *buttons,* which can be "pushed" with a mouse click

The mouse is the primary input device used with GUIs, so GUIs are sometimes called *point-and-click interfaces*. The screen shot in Figure 1.2 shows an example of a GUI.

The interface to an application or operating system is an important part of the software because it is the only part of the program the user directly interacts with. To the user, the interface *is* the program.

key concept
As far as the user is concerned, the interface *is* the program.

The focus of this book is high-quality application programs. We explore how to design and write software that will perform calculations, make decisions, and control graphics. We use the Java programming language throughout the text to demonstrate computing concepts.

digital computers

Two techniques are used to store and manage information: analog and digital. *Analog* information is continuous. For example, a thermometer is an analog device for measuring temperature. The mercury rises in a tube at the same time the temperature outside the tube rises. Another example of analog

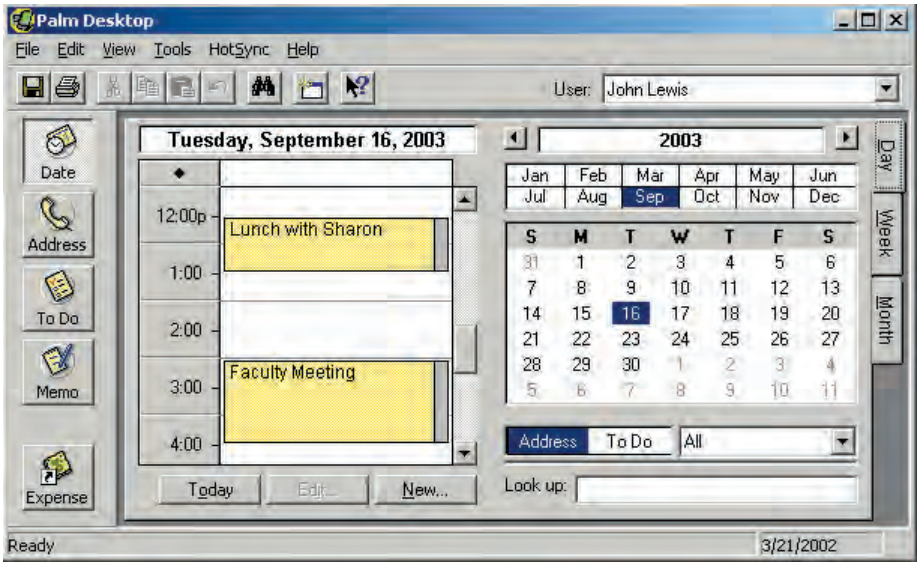


figure 1.2 An example of a graphical user interface (GUI) (Palm Desktop™ courtesy of 3COM Corporation)

information is the speed at which a car is going. As you press and release the gas and brake pedals, the car's speed varies. Figure 1.3 graphically depicts a car's speed as it varies over time.

Digital technology breaks information into pieces and shows those pieces as numbers. The music on a compact disc is stored digitally, as a series of numbers. Each number represents the voltage level of one specific instance of the recording. Many of these measurements are taken in a short period

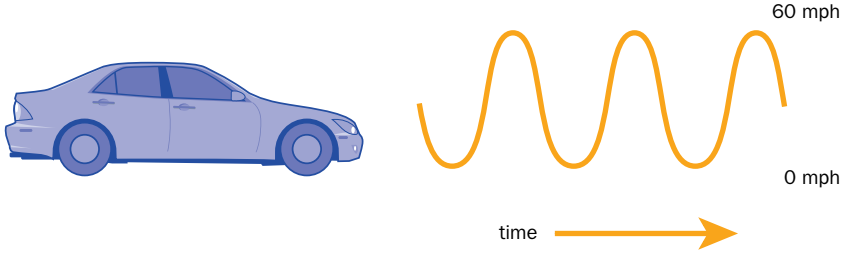


figure 1.3 A car's speed as it changes over time

of time, perhaps 40,000 measurements every second. The number of measurements per second is called the *sampling rate*. If samples are taken often enough, the separate voltage measurements can be used to create an analog signal that is "close enough" to the original. In most cases, the reproduction is good enough to satisfy the human ear.

key concept Digital computers store information by breaking it into pieces and representing each piece as a number.

Figure 1.4 shows the sampling of an analog signal. When analog information is converted to a digital format by breaking it into pieces, we say it has been *digitized*. Because the changes that occur in a signal between samples are lost, the sampling rate must be fast enough to make up the difference.

Sampling is only one way to digitize information. For example, a sentence can be stored on a computer as a series of numbers, where each number represents a single character in the sentence. Every letter, digit, and punctuation mark has been given a number. Even the space character gets a number. Consider the following sentence:

Hi, Heather.

The characters of the sentence are represented as a series of 12 numbers, as shown in Figure 1.5. When a character is repeated, such as the uppercase `'H'`, the same number is used. Note that the uppercase version of a letter is stored as a different number from the lowercase version, such as the `'H'` and `'h'` in the word Heather. They are considered different characters.

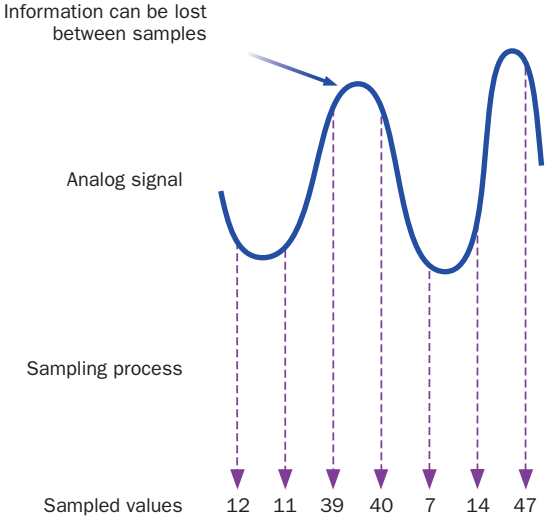


figure 1.4 Digitizing an analog signal by sampling

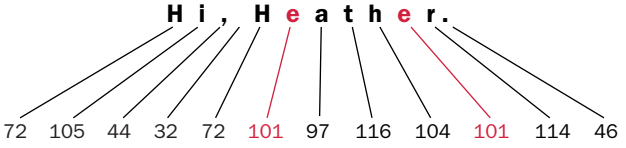


figure 1.5 Text is stored by mapping each character to a number

Modern computers are digital. Every kind of information, including text, images, numbers, audio, video, and even program instructions, is broken into pieces. Each piece is represented as a number. The information is stored by storing those numbers.

binary numbers

A digital computer stores information as numbers, but those numbers are not stored as *decimal* numbers. All information in a computer is stored and managed as *binary* numbers. Unlike the decimal system, which has 10 digits (0 through 9), the binary number system has only two digits (0 and 1). A single binary digit is called a *bit*.

All number systems work according to the same rules. The *base value* of a number system tells us how many digits we have to work with and what is the place value of each digit in a number. The decimal number system is base 10, whereas the binary number system is base 2.

Modern computers use binary numbers because the devices that store and move information are less expensive and more reliable if they have to represent only one of two possible values. Other than this, there is nothing special about the binary number system. Some computers use other number systems to store information, but they aren't as convenient.

key concept
Binary values are used to store all information in a computer because the devices that use binary information are inexpensive and reliable.

Some computer memory devices, such as hard drives, are magnetic. Magnetic material can be polarized easily to one extreme or the other, but in-between levels are hard to tell apart. So magnetic devices can be used to represent binary values very well—a magnetized area represents a binary 1 and a demagnetized area represents a binary 0. Other computer memory devices are made up of tiny electrical circuits. These devices are easier to create and are less likely to fail if they have to switch between only two states. We're better off making millions of these simple devices than creating fewer, more complicated ones.

Binary values and digital electronic signals go hand in hand. They improve our ability to send information reliably along a wire. As we've seen, an analog signal has continuously varying voltage, but a digital signal is *discrete,* which means the voltage changes dramatically between one extreme (such as +5 volts) and the other (such as –5 volts). At any point, the voltage of a digital signal is considered to be either "high," which represents a binary 1, or "low," which represents a binary 0. Figure 1.6 compares these two types of signals.

As a signal moves down a wire, it gets weaker. That is, the voltage levels of the original signal change slightly. The trouble with an analog signal is that as it changes, it loses its original information. Since the information is directly analogous to the signal, any change in the signal changes the information. The changes in an analog signal cannot be recovered because the new, degraded signal is just as valid as the original. A digital signal degrades just as an analog signal does, but because the digital signal is originally at one of two extremes, it can be reinforced before any information is lost. The voltage may change slightly from its original value, but it still can be interpreted as either high or low.

The number of bits we use in any given situation determines how many items we can represent. A single bit has two possible values, 0 and 1, so it can represent two items or situations. If we want to represent the state of a lightbulb (off or on), one bit will suffice, because we can interpret 0 as the lightbulb being off and 1 as the lightbulb being on. If we want to represent more than two things, we need more than one bit.

Two bits, taken together, can represent four items because there are exactly four ways we can arrange two bits: 00, 01, 10, and 11. Suppose we want to represent the gear that a car is in (park, drive, reverse, or neutral). We would need only two bits, and could set up a mapping between the bits and the gears. For instance, we could say that 00 represents park, 01 represents drive, 10 represents reverse, and 11 represents neutral. (Remember that

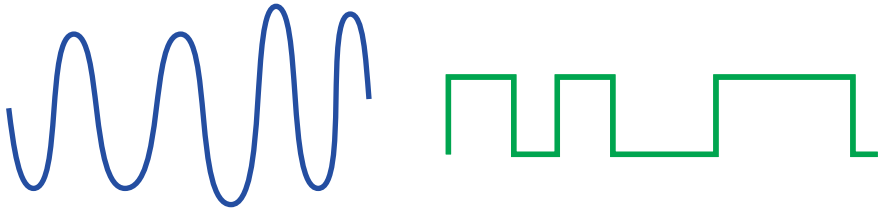


figure 1.6 An analog signal and a digital signal

'10' is not 'ten' but 'one-zero' and '11' is not 'eleven' but 'one-one.') In this case, it wouldn't matter if we switched that mapping around, though in some cases the relationships between the bit arrangements and what they represent is important.

key concept

There are exactly 2^N ways to arrange N bits. Therefore N bits can represent up to 2^N unique items.

Three bits can represent eight unique items, because there are eight arrangements of three bits. Similarly, four bits can represent 16 items, five bits can represent 32 items, and so on. Figure 1.7 shows the relationship between the number of bits used and the number of items they can represent. In general, N bits can represent 2^N unique items. For every bit added, the number of items that can be represented doubles.

We've seen how a sentence of text is stored on a computer as numeric values. Those numeric values are stored as binary numbers. Suppose we had character strings in a language with 256 characters and symbols. We would need to use eight bits to store each character because there are 256 unique ways of arranging eight bits (2^8 equals 256). Each arrangement of bits is a specific character.

Ultimately, representing information on a computer boils down to the number of items and how those items are mapped to binary values.

1 bit 2 items	2 bits 4 items	3 bits 8 items	4 bits 16 items	5 bits 32 items	
0	00	000	0000	00000	10000
1	01	001	0001	00001	10001
	10	010	0010	00010	10010
	11	011	0011	00011	10011
		100	0100	00100	10100
		101	0101	00101	10101
		110	0110	00110	10110
		111	0111	00111	10111
			1000	01000	11000
			1001	01001	11001
			1010	01010	11010
			1011	01011	11011
			1100	01100	11100
			1101	01101	11101
			1110	01110	11110
			1111	01111	11111

figure 1.7 The number of bits used determines the number of items that can be represented

1.1 hardware components

Let's look at the hardware components of a computer system in more detail. Consider the computer described in Figure 1.8. What does it all mean? Can the system run the software you want it to? How does it compare to other systems? These terms are explained in this section.

computer architecture

The architecture of a house describes its structure. Similarly, we use the term *computer architecture* to describe how the hardware components of a computer are put together. Figure 1.9 shows the basic architecture of a computer system. Information travels between components across a group of wires called a *bus*.

The CPU and the main memory make up the core of a computer. As we mentioned earlier, main memory stores programs and data that are being used, and the CPU executes program instructions one at a time.

key concept

The core of a computer is made up of the CPU and the main memory. Main memory is used to store programs and data. The CPU executes a program's instructions one at a time.

Suppose we have a program that figures out the average of a list of numbers. The program and the numbers must be in main memory while the program runs. The CPU reads one program instruction from main memory and executes it. When it needs data, such as a number in the list, the CPU reads that information as well. This process repeats until the program ends. The answer is stored in main memory to await further processing or in long-term storage in secondary memory.

Almost all devices in a computer system other than the CPU and main memory are called *peripherals*. Peripherals operate at the periphery, or outer

- 2.8 GHz Intel Pentium 4 processor
- 512 MB RAM
- 80 GB Hard Drive
- 48x CD-RW / DVD-ROM Combo Drive
- 17" Flat Screen Video Display with 1280 x 1024 resolution
- 56 Kb/s Modem

figure 1.8 The hardware specification of a particular computer

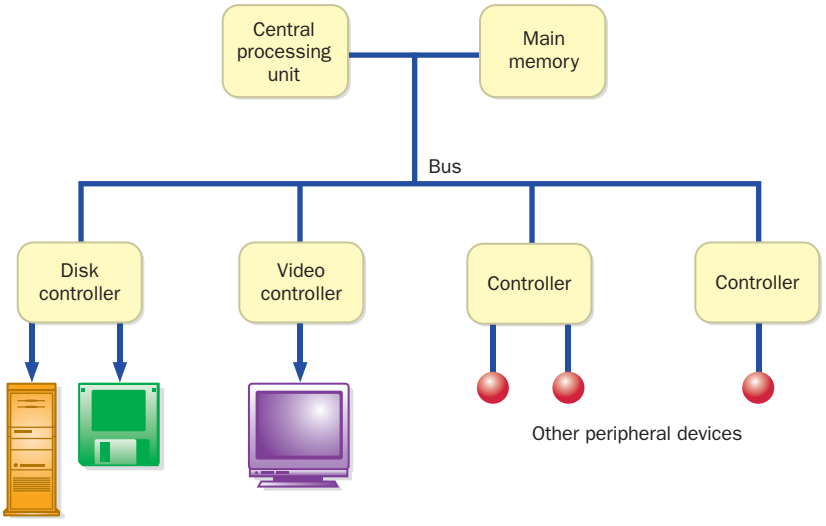


figure 1.9 Basic computer architecture

edges, of the system (although they may be in the same box). Users don't interact directly with the CPU or main memory. Instead users interact with the peripherals: the monitor, keyboard, disk drives, and so on. The CPU and main memory would not be useful without peripheral devices.

Controllers are devices that send information back and forth from the CPU and main memory to the peripherals. Every device has its own way of formatting and sending data, and part of the controller's job is to handle this. Furthermore, the controller often sends information back and forth, so the CPU can focus on other activities.

Input/output (I/O) devices and secondary memory devices are one kind of peripherals. Another kind of peripherals are *data transfer devices,* which allow information to be sent and received between computers. The computer described in Figure 1.8 has a data transfer device called a *modem,* which lets information be sent across a telephone line. The modem in the example can send data at a maximum rate of 56 *kilobits* (Kb) per second, or approximately 56,000 *bits per second* (bps).

Secondary memory devices and data transfer devices can be thought of as I/O devices because they represent a source of information (input) and a place to send information (output). For our discussion, however, we define I/O devices as devices that let the user interact with the computer.

input/output devices

Let's look at some I/O devices in more detail. The most common input devices are the keyboard and the mouse. Others include:

- *bar code readers,* such as the ones used at a grocery store checkout
- *joysticks,* often used for games and advanced graphical applications
- *microphones,* used by voice recognition systems that interpret simple voice commands
- *virtual reality devices,* such as gloves that interpret the movement of the user's hand
- *scanners,* which convert text, photographs, and graphics into machine-readable form

Monitors and printers are the most common output devices. Others include:

- *plotters,* which move pens across large sheets of paper (or vice versa)
- *speakers,* for audio output
- *goggles,* for virtual reality display

Some devices can handle both input and output. A touch screen system can detect the user touching the screen at a particular place. Software can then use the screen to display text and graphics in response to the user's touch. Touch screens are particularly useful in situations where the interface to the machine must be simple, such as at an information booth.

The computer described in Figure 1.8 includes a monitor with a 17-inch diagonal display area. A picture is created by breaking it up into small pieces called *pixels,* a term that stands for "picture elements." The monitor can display a grid of 1280 by 1024 pixels. The last section of this chapter explores the representation of graphics in more detail.

main memory and secondary memory

Main memory is made up of a series of small, connected *memory locations*, as shown in Figure 1.10. Each memory location has a unique number called an *address*.

key concept

The address is the unique number of a memory location. It is used when storing and retrieving data from memory.

When data is stored in a memory location, it overwrites and destroys any information that was stored at that location. However, data is read from a memory location without affecting it.

On many computers, each memory location consists of eight bits, or one *byte,* of information. If we need to store a value that cannot be

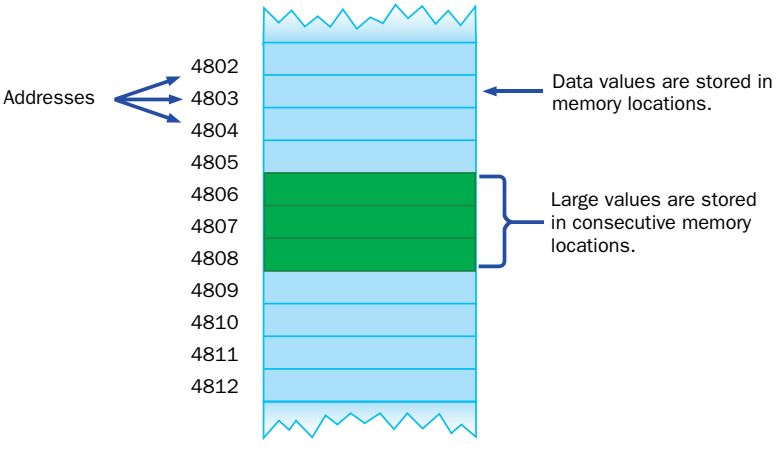


figure 1.10 Memory locations

represented in a single byte, such as a large number, then multiple, consecutive bytes are used to store the data.

key concept

Data *written* to a memory location overwrites and destroys any information that was stored at that location. Data *read* from a memory location leaves the value in memory alone.

The *storage capacity* of a device such as main memory is the total number of bytes it can hold. Devices can store thousands or millions of bytes, so you should become familiar with larger units of measure. Because computer memory is based on the binary number system, all units of storage are powers of two. A *kilobyte* (KB) is 1,024, or 2^{10}, bytes. Some larger units of storage are a *megabyte* (MB), a *gigabyte* (GB), and a *terabyte* (TB), as listed in Figure 1.11. It's usually easier to think about these numbers if we round them off. For example, most computer users think of a kilobyte as approximately one thousand bytes, a megabyte as approximately one million bytes, and so forth.

Unit	Symbol	Number of Bytes
byte		$2^0 = 1$
kilobyte	KB	$2^{10} = 1024$
megabyte	MB	$2^{20} = 1,048,576$
gigabyte	GB	$2^{30} = 1,073,741,824$
terabyte	TB	$2^{40} = 1,099,511,627,776$

figure 1.11 Units of binary storage

Many personal computers have 256 or 512 megabytes of main memory, or RAM, such as the system described in Figure 1.8. (We discuss RAM in more detail later in the chapter.) A large main memory allows large programs, or several programs, to run because they don't have to get information from secondary memory as often.

key concept

Main memory is volatile, meaning the stored information is lost when the electric power is turned off. Secondary memory devices are usually nonvolatile.

Main memory is usually *volatile,* meaning that the information stored in it will be lost if its electric power supply is turned off. When you are working on a computer, you should often save your work onto a secondary memory device such as a disk in case the power is lost. Secondary memory devices are usually *nonvolatile*, meaning the information is saved even if the power supply is turned off.

The most common secondary storage devices are hard disks and floppy disks. A high-density floppy disk can store 1.44 MB of information. The storage capacities of hard drives vary, but on personal computers, the hard drive can usually store between 40 GB and 120 GB, such as in the system described in Figure 1.8.

A disk is a magnetic medium on which bits are represented as magnetized particles. A read/write head passes over the spinning disk, reading or writing information. A hard disk drive might actually have several disks in a column with several read/write heads, such as the one shown in Figure 1.12.

To get a feel for how much information these devices can store, all the information in this book, including pictures and formatting, requires about 6 MB of storage.

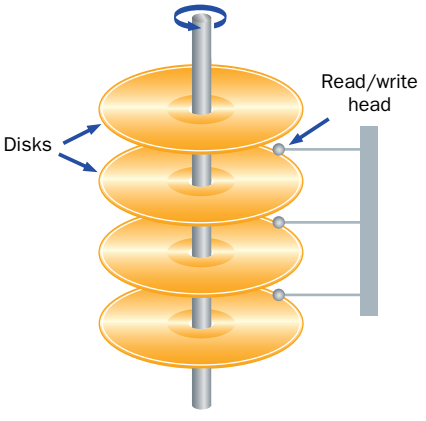


figure 1.12 A hard disk drive with multiple disks and read/write heads

Magnetic tapes are also used as secondary storage but are slower than disks because of the way information is accessed. A disk is a *direct access device* since the read/write head can move, in general, directly to the information needed. The terms *direct access* and *random access* are often confused. However, information on a tape can be accessed only after first getting past the intervening data. A tape must be rewound or fast-forwarded to get to the right place, the same way you have to fast-forward through a cassette tape to get to the song you want to hear. A tape is therefore considered a *sequential access device*. Tapes are usually used only to store information when it is no longer used very often, or to provide a backup copy of the information on a disk.

Two other terms are used to describe memory devices: *random access memory* (RAM) and *read-only memory* (ROM). It's important to understand these terms because they are used often, and their names can be misleading. RAM and main memory are basically the same thing. The term RAM seems to mean something it shouldn't. Both RAM and ROM are direct (or random) access devices. RAM should probably be called read-write memory, since data can be both written to it and read from it. Information stored on ROM, on the other hand, cannot be changed (as the term "read-only" implies). ROM chips are often embedded into the main circuit board of a computer and used to provide the instructions needed when the computer is initially turned on.

key concept

The surface of a CD has both smooth areas and small pits. A pit represents a binary 1 and a smooth area represents a binary 0.

A *CD-ROM* is a portable secondary memory device. CD stands for compact disc. It is called ROM because information is stored permanently when the CD is created and cannot be changed. Like a musical CD, a CD-ROM stores information in binary format. When the CD is created, a microscopic pit is pressed into the disc to represent a binary 1, and the disc is left smooth to represent a binary 0. The bits are read by shining a low-intensity laser beam onto the spinning disc. The laser beam reflects strongly from a smooth area on the disc but weakly from a pitted area. A sensor determines whether each bit is a 1 or a 0. A typical CD-ROM can store about 650 MB.

There are many kinds of CD technology today. It is now common for a home computer to come with a *CD-Recordable* (CD-R) drive. A CD-R can be used to create a CD for music or for general computer storage. Once created, you can use a CD-R disc in a standard CD player, but you can't change the information on a CD-R disc once it has been "burned." Music CDs that you buy in a store are pressed from a mold, whereas CD-Rs are burned with a laser.

key concept

A rewritable CD simulates the pits and smooth areas of a regular CD using a coating that can be made nonreflective or reflective as needed.

A *CD-Rewritable* (CD-RW) disc can be erased and reused. It can be reused because the pits and flat surfaces of a normal CD are made on a CD-RW by coating the surface of the disc with a material that, when heated to one temperature becomes nonreflective and when heated to a different temperature becomes reflective. The CD-RW media doesn't work in all players, but CD-Rewritable drives can create both CD-R and CD-RW discs.

CDs started as a popular format for music; they later came to be used as a general computer storage device. Similarly, the *DVD* format was first created for video and is now making headway as a general format for computer data. DVD once stood for digital video disc or digital versatile disc. A DVD has a tighter format (more bits per square inch) than a CD so it can store much more information. It is likely that DVD-ROMs will replace CD-ROMs completely because a DVD drive can read a CD-ROM. There are currently six different formats for recordable DVDs.

The speed of a CD drive is expressed in multiples of x, which represents a data transfer speed of 153,600 bytes of data per second. The drive described in Figure 1.8 has a maximum data speed of 48x, although it probably writes data at much slower speeds.

How much a device can store changes as technology improves. A general rule in the computer industry is that storage capacity doubles every 18 months. However, this progress eventually will slow down as storage capacities approach absolute physical limits.

the central processing unit

The central processing unit (CPU) uses main memory to perform all the basic processing in a computer. The CPU reads and executes instructions, one after another, in a continuous cycle. The CPU is made up of three important components, as shown in Figure 1.13. The *control unit* handles the processing steps, the *registers* are small amounts of storage space in the CPU itself, and the *arithmetic/logic unit* does calculations and makes decisions.

The control unit transfers data and instructions between main memory and the registers in the CPU. It also controls the circuitry in the arithmetic/logic unit.

In most CPUs, some registers have special purposes. For example, the *instruction register* holds the current instruction being executed. The *program counter* holds the address of the next instruction to be executed. In addition to these and other special-purpose registers, the CPU also contains a set of general-purpose registers.

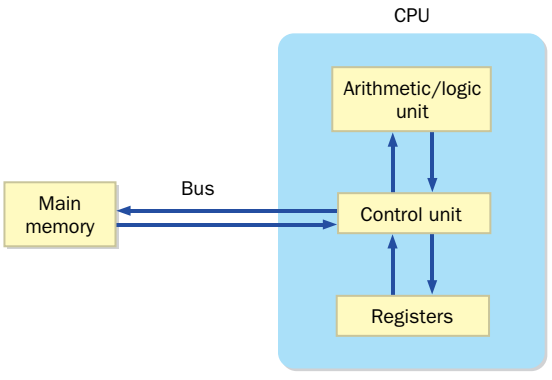


figure 1.13 CPU components and main memory

The idea of storing both program instructions and data together in main memory is called the *von Neumann architecture* of computer design, named after John von Neumann, who first advanced this programming concept in 1945. These computers continually follow the *fetch-decode-execute* cycle depicted in Figure 1.14. An instruction is fetched from main memory and put into the instruction register. The program counter increases to get ready for the next cycle. Then the instruction is decoded electronically to determine which operation to carry out. Finally, the control unit turns on the correct circuitry to carry out the instruction, which may load a data value into a register or add two values together, for example.

key concept
The von Neumann architecture and the fetch-decode-execute cycle form the foundation of computer processing.

The CPU is on a chip called a *microprocessor*, a part of the main circuit board of the computer. This board also contains ROM chips and communication sockets to which device controllers, such as the controller that manages the video display, can be connected.

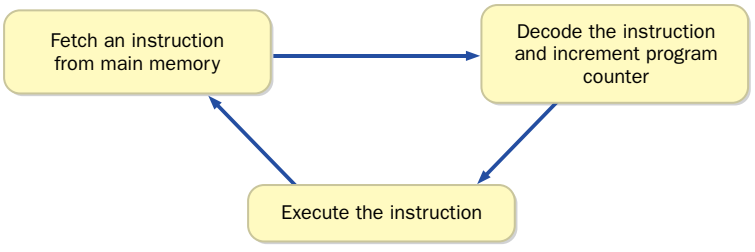


figure 1.14 The fetch-decode-execute cycle

Another part of the main circuit board is the *system clock*. The clock sends out an electronic pulse at regular intervals, so that everything going on in the CPU happens on the same schedule. The rate at which the pulses occur is called the *clock speed*, and it varies depending on the processor. The computer described in Figure 1.8 includes a Pentium 4 processor that runs at a clock speed of 2.8 gigahertz (GHz), or about 2.8 billion pulses per second. The speed of the system clock tells you about how fast the CPU executes instructions. Like storage capacities, the speed of processors is constantly increasing with advances in technology.

key concept
The speed of the system clock indicates how fast the CPU executes instructions.

1.2 networks

A single computer can do a lot, but connecting several computers together into networks can dramatically increase how much they can do and make it easier to share information. A *network* is two or more computers connected together so they can exchange information. Using networks is how commercial computers operate today. New technologies are emerging every day to improve networks.

Figure 1.15 shows a simple computer network. One of the devices on the network is a printer. Any computer connected to the network can print a document on that printer. One of the computers on the network is a *file server*, which does nothing but store programs and data that are needed by many network users. A file server usually has a large amount of secondary memory. When a network has a file server, each individual computer doesn't need its own copy of a program.

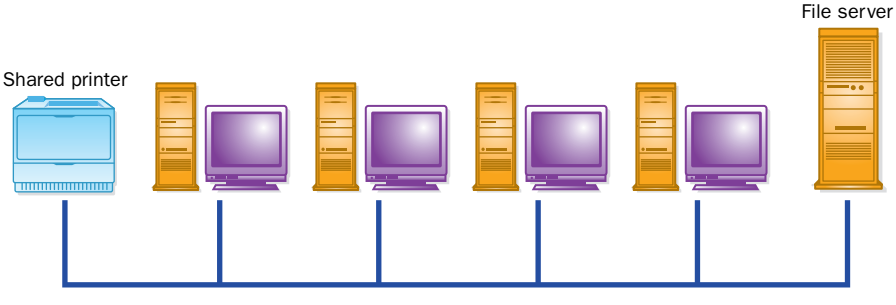


figure 1.15 A simple computer network

network connections

If two computers are directly connected, they can communicate in basically the same way that information moves across wires inside a single machine. When two computers are close to each other, this is called a *point-to-point connection*. If point-to-point connections are used, every computer is directly connected by a wire to every other computer in the network. But if the computers are far apart, having a separate wire for each connection won't work because every time a new computer is added to the network, a new wire will have to be installed for each computer already in the network. Furthermore, a single computer can handle only a small number of direct connections.

key concept
A network is two or more computers connected together so they can exchange information.

Figure 1.16 shows multiple point-to-point connections. Consider the number of wires that would be needed if two or three additional computers were added to the network.

Look at the diagrams in Figure 1.15 and Figure 1.16. All of the computers in Figure 1.15 share a single communication line. Each computer on the network has its own *network address*. These addresses are like the addresses in main memory except that they identify individual computers on a network instead of individual memory locations inside a single computer. A message from one computer to another needs the network address of the computer receiving the message.

Sharing a communication line is less expensive and makes adding new computers to the network easier. However, a shared line also means delays. The computers on the network cannot use the communication line at the same time. They have to take turns, which means they have to wait when the line is busy.

key concept
Sharing a communication line creates delays, but it is cost effective and simplifies adding new computers to the network.

One way to improve network delays is to divide large messages into small pieces, called *packets*, and then send the individual packets across the network mixed up with pieces of other messages sent by other users. The packets are collected at the destination and reassembled into the original message.

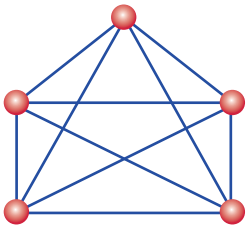


figure 1.16 Point-to-point connections

This is like a group of people using a conveyor belt to move a set of boxes from one place to another. If only one person were allowed to use the conveyor belt at a time, and that person had a lot of boxes to move, everyone else would have to wait a long time before they could use it. By taking turns, each person can put one box on at a time, and they all can get their work done. It's not as fast as having a conveyor belt of your own, but it's not as slow as having to wait until everyone else is finished.

local-area networks and wide-area networks

A *local-area network* (LAN) is designed to span short distances and connect a small number of computers. Usually a LAN connects the machines in only one building or in a single room. LANs are convenient to install and manage and are highly reliable. As computers became smaller, LANs became an inexpensive way to share information throughout an organization. However, having a LAN is like having a telephone system that allows you to call only the people in your own town. We need to be able to share information across longer distances.

key concept
A local-area network (LAN) is an inexpensive way to share information and resources throughout an organization.

A *wide-area network* (WAN) connects two or more LANs, often across long distances. Usually one computer on each LAN handles the communication across a WAN. This means the other computers in a LAN don't need to know the details of long-distance communication. Figure 1.17 shows several LANs connected into a WAN. The LANs connected by a WAN are often

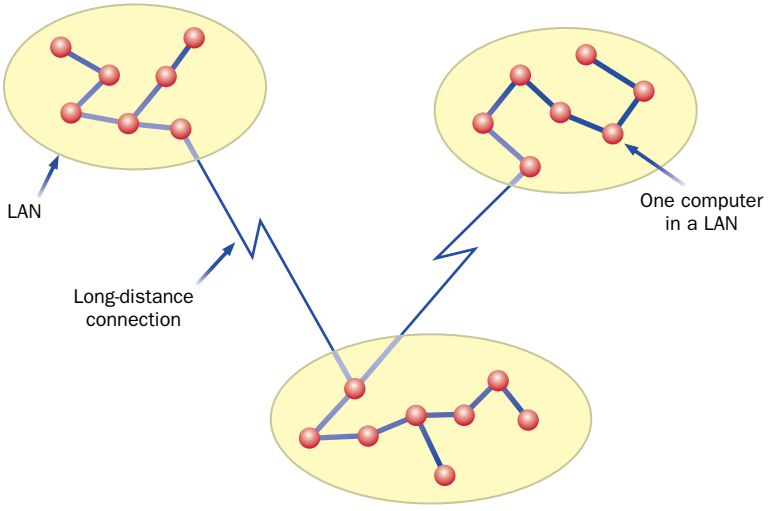


figure 1.17 LANs connected into a WAN

owned by different companies or organizations, and might even be located in different countries.

Because of networks, computing resources can now be shared among many users, and computer-based communication across the entire world is now possible. In fact, the use of networks is now so common that some computers can't work on their own.

the Internet

key concept: The Internet is a wide-area network (WAN) that spans the globe.

Throughout the 1970s, a U.S. government organization called the Advanced Research Projects Agency (ARPA) funded several projects to explore network technology. One result of these efforts was the ARPANET, a WAN that eventually became known as the Internet. The *Internet* is a network of networks. The term "Internet" comes from the word *internetworking*—connecting many smaller networks together.

From the mid 1980s through today, the Internet has grown incredibly. In 1983, there were fewer than 600 computers connected to the Internet. By the year 2000, that number had reached over 10 million. As more and more computers connect to the Internet, keeping up with the larger number of users and heavier traffic has been difficult. New technologies have replaced the ARPANET several times over, each time providing more capacity and faster processing.

key concept: TCP/IP is the set of software protocols, or rules, that govern the movement of messages across the Internet.

A *protocol* is a set of rules about how two things communicate. The software that controls the movement of messages across the Internet must follow a set of protocols called TCP/IP (pronounced by spelling out the letters, T-C-P-I-P). TCP stands for *Transmission Control Protocol*, and IP stands for *Internet Protocol*. The IP software defines how information is formatted and transferred. The TCP software handles problems such as pieces of information arriving out of order or information getting lost, which can happen if too much information arrives at one location at the same time.

Every computer connected to the Internet has an *IP address* that identifies it among all other computers on the Internet. An example of an IP address is 204.192.116.2. Fortunately, the users of the Internet rarely have to deal with IP addresses. The Internet lets each computer be given a name. Like IP addresses, the names must be unique. The Internet name of a computer is often called its *Internet address*. Two examples of Internet addresses are spencer.mcps.org and kant.gestalt-llc.com.

The first part of an Internet address is the local name of a specific computer. The rest of the address is the *domain name*. The domain name tells you

key concept
Every computer connected to the Internet has an IP address that uniquely identifies it.

about the organization to which the computer belongs. For example, mcps.org is the domain name for the network of computers in the Montgomery County public school system, and spencer might be the name of a particular computer in the network. Because the domain names are unique, many organizations can have a computer named spencer without confusion. Individual schools might be assigned *subdomains* that are added to the basic domain name. For example, the chs.mcps.org subdomain is devoted to Christiansburg High School.

The last part of each domain name, called a *top-level domain* (TLD), usually indicates the type of organization to which the computer belongs. The TLD edu indicates an educational institution. The TLD com refers to a commercial business. For example, gestalt-llc.com refers to Gestalt, LLC, a company specializing in software technologies. Another common TLD is org, used by nonprofit organizations. Many computers, especially those outside of the United States, use a TLD that tells the country of origin, such as uk for the United Kingdom. Recently, some new top-level domain names have been created, such as biz, info, and name.

When an Internet address is referenced, it gets translated to its corresponding IP address, which is used from that point on. The software that does this translation is called the *Domain Name System* (DNS). Each organization connected to the Internet operates a *domain server* that maintains a list of all computers at that organization and their IP addresses. It works like telephone directory assistance in that you give the name, and the domain server gives back a number. If the local domain server does not have the IP address for the name, it contacts another domain server that does.

The Internet has revolutionized computer processing. At first, interconnected computers were used to send electronic mail. Today the Internet connects us through the World Wide Web.

the World Wide Web

The Internet lets us share information. The *World Wide Web* (also known as WWW or simply the Web) makes sharing information easy, with the click of a mouse.

The Web is based on the ideas of hypertext and hypermedia. The term *hypertext* was first used in 1965 to describe a way to organize information. In fact, that idea was around as early as the 1940s. Researchers on the Manhattan Project, who were developing the first atomic bomb, envisioned such an approach. The idea is that documents can be linked at logical points so that the reader can jump from one document to

key concept
The World Wide Web is software that makes sharing information across a network easy.

another. When graphics, sound, animations, and video are mixed in, we call this *hypermedia*.

A *browser* is a software tool that loads and formats Web documents for viewing. *Mosaic*, the first graphical interface browser for the Web, was released in 1993. The designer of a Web document defines *links* to other Web information that might be anywhere on the Internet. Some of the people who developed Mosaic went on to found the Netscape Communications Corp. and create the popular Netscape Navigator browser, which is shown in Figure 1.18. Microsoft's Internet Explorer is another popular browser.

A computer dedicated to providing access to Web documents is called a *Web server*. Browsers load and interpret documents provided by a Web server. Many such documents are formatted using the *HyperText Markup Language* (HTML). Java programs can be embedded in HTML documents and executed through Web browsers. We explore this relationship in more detail in Chapter 2.

key concept

A browser is a software tool that loads and formats Web documents for viewing. These documents are often written using the HyperText Markup Language (HTML).

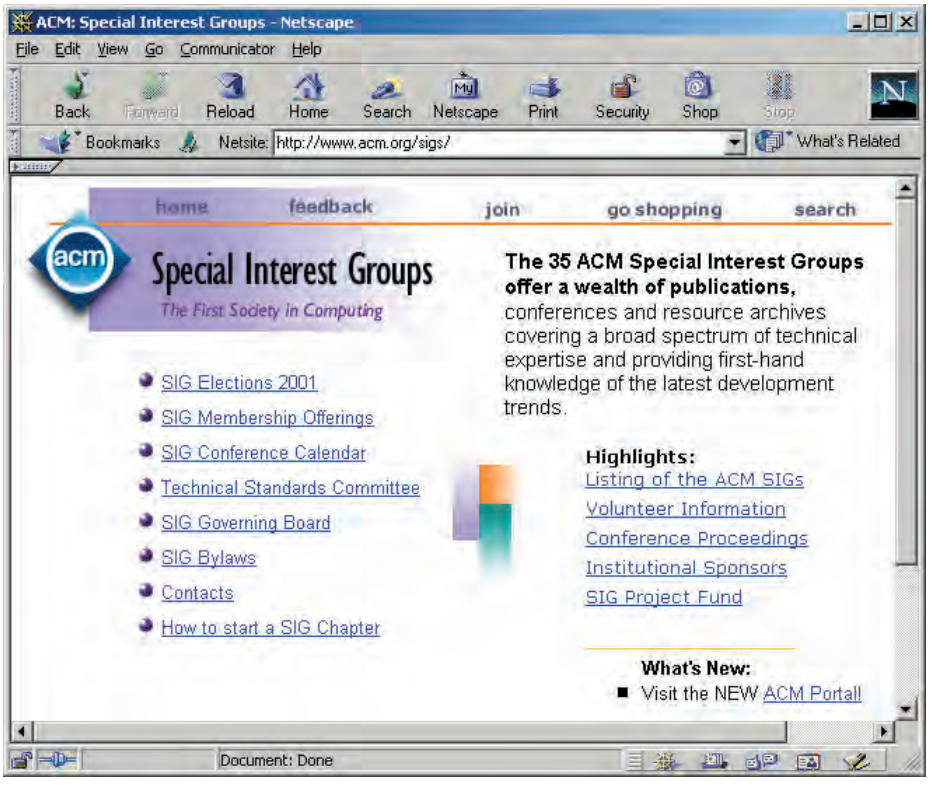


figure 1.18 Netscape Navigator browsing an HTML document (used with permission of ACM)

Uniform Resource Locators

Every Web document has a *Uniform Resource Locator* (URL). A URL uniquely specifies documents and other information for a Web browser. An example URL is:

http://www.yahoo.com

key concept
A URL uniquely specifies documents and other information found on the Web for a browser to obtain and display.

A URL contains several pieces of information. The first piece is a protocol, which determines the way the browser should communicate. The second piece is the Internet address of the machine on which the document is stored. The third piece of information is the file name. If no file name is given, as is the case with the Yahoo URL, the Web server often provides a default page (such as index.html).

Let's look at another example URL:

http://www.gestalt-llc.com/vision.html

In this URL, the protocol is http, which stands for *HyperText Transfer Protocol*. The machine referenced is www (a typical reference to a Web server), found at domain gestalt-llc.com. Finally, vision.html is a file to be transferred to the browser for viewing. Many other forms for URLs exist, but this form is the most common.

the Internet vs. the World Wide Web

The terms *Internet* and *World Wide Web* do not mean the same thing. There are important differences between the two. The Internet is a network of computers all over the world. The Web is a set of software applications that lets us use the Internet to view and exchange information. The Web is not a network. Although the Web is used effectively with the Internet, it is not bound to it. The Web can be used on a LAN that is not connected to any other network or even on a single machine to display HTML documents.

1.3 programming

The Java programming language allows software to be easily exchanged and executed via the Web. The rest of this book shows you how to create programs using Java. This section discusses the purpose of programming in general and introduces the Java programming language.

problem solving

The purpose of writing a program is to solve a problem. Problem solving, in general, consists of multiple steps:

1. Understanding the problem.
2. Breaking the problem into manageable pieces.
3. Designing a solution.
4. Considering alternatives to the solution and refining the solution.
5. Implementing the solution.
6. Testing the solution and fixing any problems.

key concept: The purpose of writing a program is to solve a problem.

Although this approach applies to any kind of problem solving, it works particularly well when developing software.

The first step, understanding the problem, may sound obvious, but skipping this step can cause us all kinds of problems. If we try to solve a problem we don't completely understand, we often end up solving the wrong problem.

After we understand the problem, we then break the problem into manageable pieces and design a solution. These steps go hand in hand. A solution to any problem is almost never one big activity. Instead, it is a series of small tasks that work together to perform a larger task. When developing software, we don't write one big program. We design separate pieces that are responsible for parts of the solution, then we put all the parts together.

key concept: The first solution we design to solve a problem may not be the best one.

Our first idea for a solution may not be the best one. We must consider all the possible solutions. The earlier we consider alternatives, the easier it is to modify our approach.

Next we take the design and put it in a usable form. This stage is where we actually write the program. Too often programming is thought of as writing code. But in most cases, this is one of the last and easiest steps. The act of designing the program should be more interesting and creative than just turning the design into a particular programming language.

Finally, we test our solution to find any mistakes so that we can fix them. Testing makes sure the program correctly represents the design, which in turn provides a solution to the problem.

Throughout this text we explore programming techniques that let us elegantly design and implement solutions to problems. Although we will often go into detail, we should not forget that programming is just a tool to help us solve problems.

the Java programming language

A program is written in a particular *programming language* that uses specific words and symbols to express the problem solution. A programming language defines a set of rules that determine exactly how a programmer can combine the words and symbols of the language into *programming statements*, which are the instructions that are carried out when the program is executed.

There are many programming languages. We use the Java language in this book to demonstrate programming concepts and techniques. Although our main goal is to learn these software development concepts, an important side-effect will be to learn the development of Java programs.

Java was developed in the early 1990s by James Gosling at Sun Microsystems. Java was introduced to the public in 1995 and has gained tremendous popularity since.

One reason Java got attention was because it was the first programming language created for the Web, but it also has important features that make it a useful general-purpose programming language.

key concept
This book focuses on the principles of object-oriented programming.

Java is an *object-oriented programming language*. Objects are the basic pieces that make up a program. Other programming languages, such as C++, let a programmer use objects but don't reinforce that approach, which can lead to confusing program designs.

Most importantly, Java is a good language to use to learn programming concepts. It doesn't get bogged down in unnecessary issues as some other languages do. Using Java, we can focus on important issues and not on less important details.

The Java language has a library of extra software that we can use when developing programs. This library lets us create graphics, communicate over networks, and interact with databases, among many other features. Although we won't be able to cover all aspects of the libraries, we will explore many of them.

Java is used all over the world. It is one of the fastest growing programming technologies of all time. So not only is it a good language in which to learn programming concepts, it is also a practical language that will serve you well in the future.

a Java program

Let's look at a simple but complete Java program. The program in Listing 1.1 prints two sentences to the screen. This program prints a quote by Abraham Lincoln. The output is shown below the program listing.

All Java applications have a similar basic structure. Despite its small size and simple purpose, this program contains several important features. Let's examine its pieces.

The first few lines of the program are comments, which start with the `//` symbols and continue to the end of the line. Comments don't affect what the program does but are included to help someone reading the code understand what the program does. Programmers should include comments throughout a program to clearly identify the purpose of the program and describe any special processing. Any written comments or documents, including a user's guide and technical references, are called *documentation*. Comments included in a program are called *inline documentation*.

key concept

Comments do not affect a program's processing; instead, they help someone reading the code understand what the program does.

listing 1.1

```
//********************************************************************
//  Lincoln.java       Author: Lewis/Loftus/Cocking
//
//  Demonstrates the basic structure of a Java application.
//********************************************************************

public class Lincoln
{
   //-----------------------------------------------------------------
   //  Prints a presidential quote.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("A quote by Abraham Lincoln:");

      System.out.println ("Whatever you are, be a good one.");
   }
}
```

output

```
A quote by Abraham Lincoln:
Whatever you are, be a good one.
```

The rest of the program in Listing 1.1 is a *class definition*. This class is called `Lincoln`, though we could have named it just about anything we wished. The class definition runs from the first opening brace (`{`) to the final closing brace (`}`) on the last line of the program. All Java programs are defined using class definitions.

Inside the class definition are some more comments describing the purpose of the `main` method, which is defined directly below the comments. A *method* is a group of programming statements that are given a name. In this case, the name of the method is `main` and it contains only two programming statements. Like a class definition, a method is also enclosed in braces.

All Java applications have a `main` method, which is where processing begins. Each programming statement in the `main` method is executed, one at a time in order, until the end of the method is reached. Then the program ends, or *terminates*. The `main` method definition in a Java program is always preceded by the words `public`, `static`, and `void`, which we examine later in the text. The use of `String` and `args` does not come into play in this particular program. We describe these later also.

key concept

In a Java application, processing begins with the `main` method. The `main` method must always be defined using the words `public`, `static`, and `void`.

The two lines of code in the `main` method invoke another method called `println` (pronounced "print line"). We *invoke*, or *call*, a method when we want it to execute. The `println` method prints the specified characters to the screen. The characters to be printed are represented as a *character string*, enclosed in double quote characters (`"`). When the program is executed, it calls the `println` method to print the first statement, calls it again to print the second statement, and then, because that is the last line in the program, the program terminates.

The code executed when the `println` method is invoked is not defined in this program. The `println` method is part of the `System.out` object, which we explore in more detail in Chapter 2.

comments

Let's look at comments in more detail. Comments are the only language feature that let programmers tell a person reading the code what they are thinking. Comments should tell the reader what the programmer wants the program to do. A program is often used for many years, and often many changes are made to it over time. Even the original programmer may not remember the details of a particular program when, at some point in the future, changes are needed. Furthermore, the original programmer is not always available to make the changes, and someone completely unfamiliar

with the program will need to understand it. Good documentation is therefore very important.

As far as the Java programming language is concerned, comments can be written using any content whatsoever. Comments are ignored by the computer; they do not affect how the program executes.

The comments in the `Lincoln` program represent one of two types of comments allowed in Java. The comments in `Lincoln` take the following form:

```
// This is a comment.
```

This type of comment begins with a double slash (`//`) and continues to the end of the line. You cannot have any characters between the two slashes. The computer ignores any text after the double slash and to the end of the line. A comment can follow code on the same line to document that particular line, as in the following example:

```
System.out.println ("Monthly Report"); // always use this title
```

The second form a Java comment may have is:

```
/*  This is another comment.  */
```

This comment type does not use the end of a line to indicate the end of the comment. Anything between the first slash-asterisk (`/*`) and the second asterisk-slash (`*/`) is part of the comment, including the invisible *newline* character that represents the end of a line. Therefore, this type of comment can extend over multiple lines. No space can be between the slash and the asterisk.

The two basic comment types can be used to create different documentation styles, such as:

```
// This is a comment on a single line.

//-------------------------------------------------------------
// Some comments such as those above methods or classes
// deserve to be blocked off to focus special
// attention on a particular aspect of your code.  Note
// that each of these lines is technically a separate
// comment.
//-------------------------------------------------------------

/*
   This is one comment
   that spans several lines.
*/
```

Programmers often concentrate so much on writing code that they focus too little on documentation. You should develop good commenting practices and make them a habit. Comments should be well written, often in complete sentences. They should not tell the reader things that are obvious or confusing. The following examples are *not* good comments:

```
System.out.println ("hello");  // prints hello
System.out.println ("test");   // change this later
```

The first comment tells the obvious purpose of the line and does not add any new information to the statement. It is better to have no comment than a useless one. The second comment is confusing. What should be changed later? When is later? Why should it be changed?

key concept
Inline documentation should not be confusing or obvious.

It is considered good programming style to use comments in a consistent way throughout an entire program.

identifiers and reserved words

The various words used when writing programs are called *identifiers*. The identifiers in the `Lincoln` program are `class`, `Lincoln`, `public`, `static`, `void`, `main`, `String`, `args`, `System`, `out`, and `println`. These fall into three categories:

- words that we make up (`Lincoln` and `args`)
- words that another programmer chose (`String`, `System`, `out`, `println`, and `main`)
- words that are reserved for special purposes in the language (`class`, `public`, `static`, and `void`)

While writing the program in Listing 1.1, we simply chose to name the class `Lincoln`, but we could have used one of many other names. For example, we could have called it `Quote`, or `Abe`, or `GoodOne`. The identifier `args` (which is short for arguments) is often used in the way we use it in `Lincoln`, but we could have used just about any identifier in its place.

The identifiers `String`, `System`, `out`, and `println` were chosen by other programmers. These words are not part of the Java language. They are part of a huge library of predefined code, a set of classes and methods that someone has already written for us. The authors of that code chose the identifiers—we're just using them. We discuss this library of predefined code in more detail in Chapter 2.

Reserved words are identifiers that have a special meaning in a programming language and can only be used in predefined ways. In the `Lincoln`

program, the reserved words used are `class`, `public`, `static`, and `void`. Throughout this book, we show Java reserved words in blue type. Figure 1.19 lists all of the Java reserved words in alphabetical order. The words marked with an asterisk are reserved for possible future use in later versions of the language but right now have no meaning in Java. A reserved word cannot be used for any other purpose, such as naming a class or method.

An identifier that we make up for use in a program can be any combination of letters, digits, the underscore character (_), and the dollar sign ($), but it cannot begin with a number. Identifiers may be of any length. Therefore `total`, `label7`, `nextStockItem`, `NUM_BOXES`, and `$amount` are all valid identifiers, but `4th_word` and `coin#value` are not valid.

Both uppercase and lowercase letters can be used in an identifier, and the difference is important. Java is *case sensitive*, which means that two identifier names that differ only in the case of their letters are considered to be different identifiers. Therefore `total`, `Total`, `ToTaL`, and `TOTAL` are all different identifiers. As you can imagine, it is not a good idea to use identifiers that differ only in their case because they can be easily confused.

Although the Java language doesn't require it, using the same case format for each kind of identifier makes your identifiers easier to understand. For example, we use *title case* (uppercase for the first letter of each word) for class names. That is a Java convention, although it does not technically have to be followed. Throughout the text, we

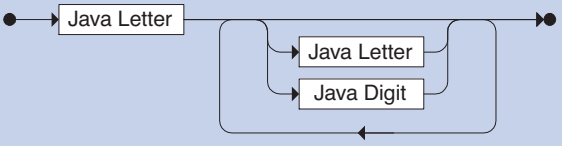
key concept

Java is case sensitive. The uppercase and lowercase versions of a letter are distinct. You should use the same case convention for different types of identifiers.

abstract	default	goto*	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const*	float	new	switch	
continue	for	null	synchronized	

figure 1.19 Java reserved words

Identifier



An identifier is a letter followed by zero or more letters and digits. A Java Letter includes the 26 letters of the English alphabet in both uppercase and lowercase, the `$` and `_` (underscore) characters, as well as alphabetic characters from other languages. A Java Digit includes the numbers `0` though `9`.

Examples:

```
total
MAX_HEIGHT
num1
Keyboard
```

describe the preferred case style for each type of identifier as we introduce them.

An identifier can be as long as you like, but you should choose your names carefully. They should be descriptive but not wordy. Don't use meaningless names such as `a` or `x`. An exception to this rule can be made if the short name is actually descriptive, such as using `x` and `y` to represent (x, y) coordinates on a graph. Likewise, you should not use unnecessarily long names, such as the identifier `theCurrentItemBeingProcessed`. The name `currentItem` would serve just as well.

key concept — Identifier names should be descriptive and readable.

As you might imagine, wordy identifiers are much less common than the names that are not descriptive. You should always be careful when abbreviating words. You might think `curStVal` is a good name to represent the current stock value, but another person trying to understand the code may have trouble figuring out what you meant. It might not even be clear to you two months after writing it.

A *name* in Java is a series of identifiers separated by the dot (period) character. The name `System.out` is the way we designate the object through which we invoked the `println` method. Names appear quite regularly in Java programs.

white space

All Java programs use *white space* to separate the words and symbols used in a program. White space consists of blanks, tabs, and newline characters. The phrase *white space* refers to the fact that, on a white sheet of paper with black printing, the space between the words and symbols is white. A programmer uses white space to emphasize parts of the code and make a program easier to read.

key concept
White space can make a program easier to read and understand.

Except when it's used to separate words, the computer ignores white space. It does not affect the execution of a program. This fact gives programmers a great deal of flexibility in how they format a program. The lines of a program should be divided in logical places and certain lines should be indented and aligned so that the program's structure is clear.

Because white space is ignored, we can write a program in many different ways. For example, we could put as many words as possible on each line. The code in Listing 1.2, the `Lincoln2` program, is formatted quite differently from Listing 1.1, `Lincoln`, but prints the same message.

Taking white space to the other extreme, we could write almost every word and symbol on a different line, such as `Lincoln3`, shown in Listing 1.3.

All three versions of `Lincoln` are technically valid and will execute in the same way, but they are different from a reader's point of view. Listings 1.2 and 1.3 show poor style and make the program hard to understand. You should use a set of style guidelines that increase the readability of your code.

key concept
You should always follow a set of guidelines that establish the way you format and document your programs.

listing 1.2

```
//********************************************************************
//  Lincoln2.java       Author: Lewis/Loftus/Cocking
//
//  Demonstrates a poorly formatted, though valid, program.
//********************************************************************

public class Lincoln2{public static void main(String[]args){
System.out.println("A quote by Abraham Lincoln:");
System.out.println("Whatever you are, be a good one.");}}
```

output

```
A quote by Abraham Lincoln:
Whatever you are, be a good one.
```

listing 1.3

```
//********************************************************************
//  Lincoln3.java       Author: Lewis/Loftus/Cocking
//
//  Demonstrates another valid program that is poorly formatted.
//********************************************************************

          public       class
     Lincoln3
   {
                public
   static
        void
  main
        (
String
            []
    args                         )
  {
  System.out.println        (
"A quote by Abraham Lincoln:"          )
  ;         System.out.println
             (
       "Whatever you are, be a good one."
      )
  ;
}
          }
```

output

```
A quote by Abraham Lincoln:
Whatever you are, be a good one.
```

1.4 programming languages

Suppose you are giving travel directions to a friend. You might explain those directions in any one of several languages, such as English, French, or Italian. The directions are the same no matter which language you use, but the way you express the directions is different. Furthermore, your friend must be able to understand the language you use in order to follow the directions.

Similarly, you can write a program in one of many programming languages, such as Java, Ada, C, C++, Pascal, and Smalltalk. The purpose of the program is the same no matter which language you use, but the particular

statements used to express the instructions, and the overall organization of those instructions, vary with each language. Furthermore, a computer must be able to understand the instructions in order to carry them out.

This section explores types of programming languages and describes the special programs used to prepare and execute them.

programming language levels

There are four groups of programming languages. These groups basically reflect the historical development of computer languages:

- machine language
- assembly language
- high-level languages
- fourth-generation languages

In order for a program to run on a computer, it must be in that computer's *machine language*. Each type of CPU has its own language. For that reason, we can't run a program written for a Sun Workstation, with its Sparc processor, on an IBM PC, with its Intel processor.

Each machine language instruction can do only a simple task. For example, a single machine language instruction might copy a value into a register or compare a value to zero. It might take four separate machine language instructions to add two numbers together and to store the result. However, a computer can do millions of these instructions in a second, and therefore many simple commands can be quickly executed to accomplish complex tasks.

key concept
All programs must be translated to a particular CPU's machine language in order to be executed.

Machine language code is expressed as a series of binary digits and is extremely difficult for humans to read and write. Originally, programs were entered into the computer using switches or some similarly tedious method. These techniques were time consuming and error prone.

Next came *assembly language,* which replaced binary digits with *mnemonics,* short English-like words that represent commands or data. It is much easier for programmers to deal with words than with binary digits. However, an assembly language program cannot be executed directly on a computer. It must first be translated into machine language.

Generally, each assembly language instruction equals a machine language instruction. Therefore, like machine language, each assembly language instruction does only one simple operation. Although assembly language is better than machine code from a programmer's point of view, it is still

tedious to use. Both assembly language and machine language are considered *low-level languages*.

Today, most programmers use a *high-level language* to write software. A high-level language uses English-like phrases, so it is easier for programmers to read and write. A single high-level language programming statement can accomplish the equivalent of many—perhaps hundreds—of machine language instructions. The term *high-level* means the programming statements are like natural language. Java is a high-level language, as are Ada, C, C++, and Smalltalk.

Figure 1.20 shows the same expressions written in a high-level language, assembly language, and machine language. The expressions add two numbers together.

The high-level language expression in Figure 1.20 is readable for programmers. It is like an algebraic expression. The same assembly language code is longer and somewhat less readable. The machine language is basically unreadable and much longer. In fact, only a small portion of the binary machine code to add two numbers together is shown in Figure 1.20. The complete machine language code for this particular expression is over 400 bits long.

High-level language code must be translated into machine language before it can be executed. A high-level language means programmers don't need to know the machine language for the processor on which they are working.

key concept

Working with high-level languages lets the programmer ignore the details of machine language.

Some programming languages operate at an even higher level than high-level languages. They might be used for automatic report generation or interaction with a database. These languages are called *fourth-generation languages*, or simply 4GLs, because they followed the first three generations of computer programming: machine, assembly, and high-level.

High-Level Language	Assembly Language	Machine Language

a + b	ld [%fp–20], %o0
	ld [%fp–24], %o1
	add %o0, %o1, %o0
		...
		1101 0000 0000 0111
		1011 1111 1110 1000
		1101 0010 0000 0111
		1011 1111 1110 1000
		1001 0000 0000 0000
		...

figure 1.20 The same expression in a high-level language, assembly language, and machine language

compilers and interpreters

Several special-purpose programs are needed to help with the process of developing new programs. They are sometimes called *software tools* because they are used to build programs. Examples of basic software tools include an editor, a compiler, and an interpreter.

You use an *editor* as you type a program into a computer and store it in a file. There are many different editors with many different features. You should get to know the editor you will use regularly so you can enter and change your programs quickly.

Each time you need to make a change to the code of your program, you open it in an editor. Figure 1.21 shows a very basic view of the program development process. After editing and saving your program, you try to translate it from high-level code into a form that can be executed. That translation may result in errors, in which case you return to the editor to make changes to the code to fix the problems. Once the translation works, you can execute the program and see the results. If the results are not what you want, you again return to the editor to make changes.

The translation of source code into (ultimately) machine language for a particular type of CPU can occur in many ways. A *compiler* is a program that translates code in one language to code in another language. The original code is called *source code*, and the language into which it is translated is called the *target language*. For many compilers, the source code is translated directly into a particular machine language. In that case, the translation process occurs once, and the resulting program can be run whenever needed.

An *interpreter* is like a compiler but with an important difference. An interpreter does the translation and execution in short bursts. A small part of the source code, such as one statement, is translated and executed. Then another statement is translated and executed, and so on. One advantage of this technique is that it eliminates the need for a separate compilation phase. However, the program generally runs more slowly because the translation process occurs during each execution.

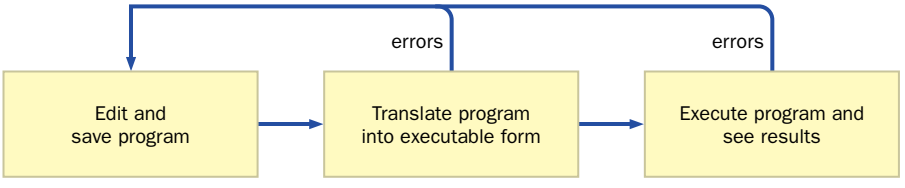


figure 1.21 Editing and running a program

key concept

A Java compiler translates Java source code into Java bytecode. A Java interpreter translates and executes the bytecode.

The process often used to translate and execute Java programs combines the use of a compiler and an interpreter. This process is pictured in Figure 1.22. The Java compiler translates Java source code into Java *bytecode,* which is a low-level form something like machine language code. The Java interpreter reads Java bytecode and executes it on a specific machine. Another compiler could translate the bytecode into a particular machine language for execution on that machine.

The difference between Java bytecode and true machine language code is that Java bytecode is not tied to any particular processor type. This makes Java *architecture neutral,* and therefore will work on many types of machines. The only restriction is that there must be a Java interpreter or a bytecode compiler for each processor type on which the Java bytecode is to be executed.

Since the compilation process translates the high-level Java source code into a low-level representation, the interpretation process works better than interpreting high-level code directly. Executing a program by interpreting its bytecode is still slower than executing machine code directly, but it is fast enough for most applications. Note that Java bytecode could be compiled into machine code.

The Java compiler and interpreter are part of the Java *Software Development Kit* (SDK), which is sometimes referred to simply as the *Java*

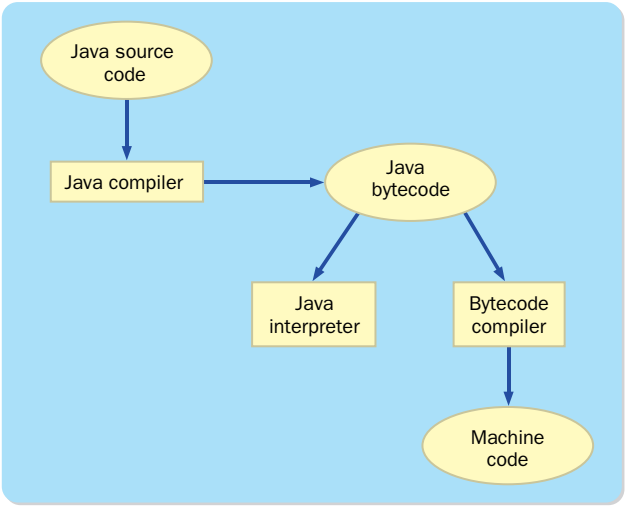


figure 1.22 The Java translation and execution process

Development Kit (JDK). This kit also contains several other software tools that may be useful to a programmer. The JDK can be downloaded for free from the Sun Microsystem Web site (java.sun.com) or from this book's Web site. Note that the standard JDK tools are executed on the command line. That is, they are not graphical programs with menus and buttons but rather are used by typing commands in a command window.

key concept

Java is architecture neutral because Java bytecode is not associated with any particular machine.

Other programs, called *Integrated Development Environments* (IDEs), support the development of Java programs. IDEs combine an editor, compiler, and other Java support tools. Sun has a Java IDE called NetBeans (www.netbeans.org) that incorporates the development tools of the JDK into one convenient GUI-based program. IBM promotes a similar IDE called Eclipse (www.eclipse.org). Both NetBeans and Eclipse are *open source* projects, meaning that they are developed by many different programmers and are available for free. Which tools you will use to develop your programs depend on your environment.

syntax and semantics

Each programming language has its own unique *syntax*. The syntax rules of a language dictate exactly how the vocabulary elements of the language can be combined to form statements. These rules must be followed in order to create a program. We've already discussed several Java syntax rules (for instance, the fact that an identifier cannot begin with a number is a syntax rule). The fact that braces are used to begin and end classes and methods is also a syntax rule. During compilation, all syntax rules are checked. If a program is not syntactically correct, the compiler will issue error messages and will not produce bytecode.

The *semantics* of a statement in a programming language define what will happen when that statement is executed. The semantics of a program are usually very well defined. That is, there is one and only one interpretation for each statement. On the other hand, the language that people use, such as English or French, can often have two or more different meanings. For example, consider the following sentence:

Time flies like an arrow.

Most people would take this sentence to mean that time moves quickly in the same way that an arrow moves quickly. However, if *time* is a verb (as in "run the 50-yard dash and I'll time you") and the word *flies* is a noun (the plural

key concept
The syntax rules of a programming language dictate the form of a program. The semantics dictate the meaning of the program statements.

of fly), the meaning changes completely. A computer would have a hard time determining which meaning is correct. Moreover, this statement could describe the preferences of an unusual insect known as a "time fly," which might be found near an archery range. After all, as Groucho Marx pointed out, fruit flies like a banana.

The point is that English allows multiple valid meanings, but a computer language can't. If a programming language instruction could have two different meanings, a computer would not be able to tell which one to follow.

errors

Several different kinds of problems can occur in software, particularly during program development. The term *computer error* is often misused. From a user's point of view, anything that goes wrong is often called a computer error. For example, suppose you charged a $23 item to your credit card, but when you received the bill, the item was listed at $230. After you have the problem fixed, the credit card company apologizes for the "computer error." Did the computer arbitrarily add a zero to the end of the number, or did it perhaps multiply the value by 10? Of course not. A computer does what we tell it to do and uses the data we give it. If our programs or data are wrong, then we can't expect the results to be correct. We call this "garbage in, garbage out."

key concept
A computer follows our instructions exactly. The programmer is responsible for the accuracy and reliability of a program.

You will encounter three kinds of errors as you develop programs:

- compile-time error
- runtime error
- logical error

The compiler checks to make sure you are using the correct syntax. If the syntax is wrong the compiler will produce a *syntax error.* The compiler also tries to find other problems, such as the use of incompatible types of data. The syntax might be technically correct, but you are still attempting to do something that the language doesn't semantically allow. Any error identified by the compiler is called a *compile-time error.* If a compile-time error occurs, an executable version of the program is not created.

key concept
A Java program must be syntactically correct or the compiler will not produce bytecode.

The second kind of problem occurs during program execution. It is called a *runtime error,* and it causes the program to terminate abnormally or "crash." For example, if we try to divide by zero, the program will crash. The system simply stops processing your program. The best programs are *robust;* that is, they avoid as many runtime errors as possible. For example, the program code could guard against the possibility of dividing by zero and handle the situation appropriately if it arises. In Java, many runtime errors are represented as *exceptions* that can be caught and dealt with. We discuss exceptions in Chapter 5.

The third kind of software problem is a *logical error.* In this case, the software compiles and executes without complaint, but it produces the wrong results. For example, a logical error occurs when a value is calculated incorrectly, such as adding two numbers when they should have been multiplied. A programmer must test the program thoroughly, comparing the expected results to those that actually occur. When defects are found, they must be traced back to the source of the problem in the code and corrected. Finding and correcting defects in a program is called *debugging*. Logical errors can show up in many ways, and the cause might be hard to find.

language evolution

As computer technology evolves, so must the languages we use to program them. The Java programming language has changed since its creation. This text uses the most recent Java technology. Specifically, this book uses the *Java 2 Platform,* which simply refers to the most advanced collection of Java language features, software libraries, and tools. Several important changes have been made since the previous version. The Java 2 Platform is organized into three major groups:

- Java 2 Platform, Standard Edition (J2SE)
- Java 2 Platform, Enterprise Edition (J2EE)
- Java 2 Platform, Micro Edition (J2ME)

This book focuses on the Standard Edition, which, as the name implies, is the mainstream version of the language and associated tools. Furthermore, this book is based on the most recent version of the Standard Edition, which is J2SE 5.0.

GRAPHICS TRACK

1.5 graphics

Graphics play an important role in computer systems. In this book we explore graphics and discuss how they are created and used. In fact, the last one or two sections of each chapter are devoted to graphics topics. (These sections can be skipped without losing continuity through the rest of the text.) In this section, we explore representing a picture in a computer and displaying it on a screen.

A picture, like all other information stored on a computer, must be digitized by breaking the information into pieces and representing those pieces as numbers. In the case of pictures, we break the picture into *pixels* (picture elements), as we mentioned earlier in this chapter. A pixel is a very small piece of the picture. The complete picture is stored by storing the color of each pixel.

key concept: The pixels of a black-and-white picture can be represented using a single bit each, 0 for white and 1 for black.

A black-and-white picture can be stored by representing each pixel using a single bit. If the bit is zero, that pixel is white; if the bit is 1, it is black. The more pixels used to represent a picture, the more realistic it looks. Figure 1.23 shows a black-and-white picture that has been stored digitally and an enlargement of part of that picture, which shows the pixels.

figure 1.23 A digitized picture with a small portion magnified

coordinate systems

When drawn, each pixel of a picture is mapped to a pixel on the screen. Each computer system and programming language defines a coordinate system like a coordinate system on a street map, so that we can find particular pixels.

The Java programming language has a relatively simple coordinate system. Figure 1.24 shows the Java coordinate system.

Each point in the Java coordinate system is represented using an (x, y) pair of values. The top-left corner of any Java drawing area has coordinates (0, 0). The x-axis coordinates get larger as you move to the right, and the y-axis coordinates get larger as you move down.

A Java program does not have to be graphical in nature. However, if it is, each graphical component in the program has its own coordinate system, with the origin (0, 0) in the top-left corner. This makes it easy to manage graphical elements.

representing color

Color pictures are divided into pixels, just as black-and-white pictures are. However, because each pixel can be one of many colors, it is not enough to represent each pixel using only one bit. There are many ways to represent the color of a pixel. This section explores one popular technique.

key concept

The pixels of a color picture can be represented using three numbers, collectively called the RGB value, which represent the relative contributions of three primary colors: red, green, and blue.

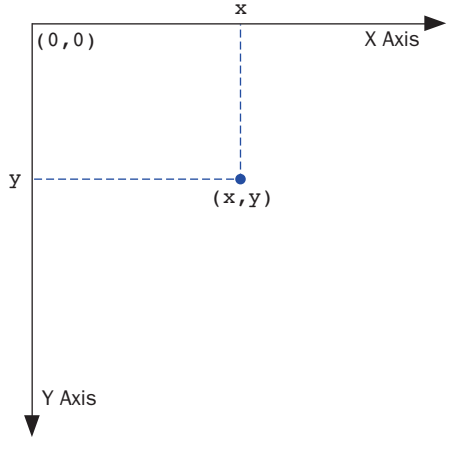


figure 1.24 The Java coordinate system

Every color can be represented as a mix of three *primary colors:* red, green, and blue. In Java, as in many other computer languages, colors are specified by three numbers called an *RGB value*. RGB stands for Red-Green-Blue. Each number represents the contribution of a primary color. Using one byte (8 bits) to store each of the three numbers, the numbers can range from 0 to 255. How much of each primary color determines the overall color. For example, high values of red and green combined with a low level of blue results in a shade of yellow.

In the graphics sections of other chapters we explore the use of color and how to control it in a Java program.

summary of key concepts

- A computer system consists of hardware and software that work together to help us solve problems.
- To execute a program, the computer first copies the program from secondary memory to main memory. The CPU then reads the program instructions from main memory, executing them one at a time until the program ends.
- The operating system provides a user interface and manages computer resources.
- As far as the user is concerned, the interface *is* the program.
- Digital computers store information by breaking it into pieces and representing each piece as a number.
- Binary values are used to store all information in a computer because binary-based devices are inexpensive and reliable.
- There are exactly 2^N ways of arranging N bits. Therefore N bits can represent up to 2^N unique items.
- The core of a computer is made up of the CPU and the main memory. Main memory is used to store programs and data. The CPU executes a program's instructions one at a time.
- An address is a unique number assigned to each memory location. It is used when storing and retrieving data from memory.
- Data *written* to a memory location overwrites and destroys any information that was stored at that location. Data *read* from a memory location leaves information in memory alone.
- The information in main memory is stored only as long as electric power is supplied. The information in secondary memory is stored until it is deliberately deleted.
- The surface of a CD has both smooth areas and small pits. A pit represents a binary 1 and a smooth area represents a binary 0.
- A rewritable CD imitates the pits and smooth areas of a regular CD using a coating that can be made nonreflective or reflective as needed.
- The von Neumann architecture and the fetch-decode-execute cycle are the foundation of computer processing.
- The speed of the system clock tells us how fast the CPU executes instructions.

- A network is two or more computers connected together so they can exchange information.
- Sharing a communication line creates delays, but it is inexpensive and makes adding new computers to the network easier.
- A local-area network (LAN) is an inexpensive way to share information and resources throughout an organization.
- The Internet is a wide-area network (WAN) that spans the globe.
- TCP/IP is the set of software protocols, or rules, for moving messages across the Internet.
- Every computer connected to the Internet has a unique IP address.
- The World Wide Web is software that makes sharing information across a network easy.
- A browser is a software tool that loads and formats Web documents for viewing. These documents are often written in HyperText Markup Language (HTML).
- A URL is the unique name of a Web document that a browser needs to find and display it.
- The purpose of writing a program is to solve a problem.
- The first solution to a problem may not be the best one.
- Java is an object-oriented programming language.
- Comments help the people who read code understand what the programmer had in mind.
- Comments should be clear and helpful.
- In a Java application, processing begins with the `main` method. The `main` method must always be defined using the words `public`, `static`, and `void`.
- Java is case sensitive. The uppercase and lowercase versions of a letter are distinct. You should use the same case convention for different types of identifiers.
- Identifier names should be descriptive and readable.
- White space can make a program easier to read and understand.
- You should always follow a set of guidelines that establish the way you format and document your programs.
- All programs must be translated to a particular CPU's machine language in order to be executed.

- Working with high-level languages lets the programmer ignore the machine language.
- A Java compiler translates Java source code into Java bytecode. A Java interpreter translates and executes the bytecode.
- Java is architecture neutral because Java bytecode doesn't have to run on any particular hardware platform.
- The syntax rules of a programming language dictate the form of a program. The semantics dictate the meaning of the program statements.
- A computer follows our instructions exactly. The programmer is responsible for the accuracy and reliability of a program.
- A Java program must be syntactically correct or the compiler will not produce bytecode.
- The pixels of a black-and-white picture can be represented using a single bit each, 0 for white and 1 for black.
- The pixels of a color picture can be represented using three numbers, called the RGB value, for the three primary colors: red, green, and blue.

self-review questions

1.1 What is hardware? What is software?

1.2 What are the two jobs of an operating system?

1.3 What happens to information when it is stored digitally?

1.4 How many items can be represented with the following?

a. 2 bits

b. 4 bits

c. 5 bits

d. 7 bits

1.5 How many bits are there in each of the following?

a. 8 bytes

b. 2 KB

c. 4 MB

1.6 What are the two main hardware components in a computer? How do they work with each other?

1.7 What is a memory address?

1.8 What does volatile mean? Which memory devices are volatile and which are nonvolatile?

1.9 What is a file server?

1.10 What is the total number of communication lines needed for a fully connected point-to-point network of five computers? Six computers?

1.11 Where does the word "Internet" come from?

1.12 Explain the parts of the following URLs:

a. chs.mcps.org/Faculty/math.htm

b. java.sun.com/products/index.html

1.13 What is the difference between a high-level language and machine language?

1.14 What is Java bytecode?

1.15 What is white space? Does it change program execution?

1.16 Which of the following are not valid Java identifiers? Why?

a. `RESULT`

b. `result`

c. `12345`

d. `x12345y`

e. `black&white`

f. `answer_7`

1.17 What do we mean by the syntax and semantics of a programming language?

1.18 How can a black-and-white picture be represented using 1s and 0s?

multiple choice

1.1 If a picture was made up of 64 possible colors, how many bits would be needed to store each pixel of the picture?

a. 4

b. 5

c. 6

d. 7

e. 8

1.2 How many bits are there in 12 KB?

a. 12,000

b. 98,304

c. 8192

d. 9600

e. 12,288

1.3 Which of the following is equivalent to $2^{20} \times 2^2$ bits?

a. 2 KB

b. 4 KB

c. 2 MB

d. 4 MB

e. 4 GB

1.4 How many different items can be represented with 11 bits?

a. 11

b. 22

c. 121

d. 1100

e. 2048

1.5 Which of the following is an example of an analog device?

a. mercury thermometer

b. computer

c. music CD

d. digital alarm clock

e. vending machine

1.6 Which of the following is *not* a valid Java identifier?

a. `Factorial`

b. `anExtremelyLongIdentifierIfYouAskMe`

c. `2ndLevel`

d. `level2`

e. `highest$`

1.7 Which of the following *is* a valid Java identifier?

a. `14andCounting`

b. `max_value`

c. `123`

d. `%taxRate`

e. `hook&ladder`

1.8 Which of the following pairs of variables are different from each other?

a. `Total` and `total`

b. `case` and `CASE`

c. `codeTwo` and `code2`

d. `oneMore` and `one_More`

e. all of the above

true/false

1.1 The identifiers `Maximum` and `maximum` are considered the same in Java.

1.2 ROM means random access device.

1.3 Computers continually follow the fetch-decode-execute cycle.

1.4 A network is two or more computers connected together so they can exchange information.

1.5 The first step in problem solving is to start implementing the solution.

1.6 Web pages are usually formatted using the HyperText Markup Language (HTML).

1.7 Identifiers in Java may contain any characters you can find on your keyboard.

1.8 Java is an object-oriented programming language.

1.9 The term *white space* refers to characters that are not part of the alphabet or numbers, such as the symbols %, &, and @.

1.10 Java is an assembly language.

short answer

1.1 Describe the hardware parts of your personal computer or of a computer in your school lab. Include the processor type and speed, storage capacities of main and secondary memory, and types of I/O devices.

1.2 Why do we use the binary number system to store information on a computer?

1.3 If a language uses 240 letters and symbols, how many bits would be needed to store each character of a document? Why?

1.4 Explain the difference between random access memory (RAM) and read-only memory (ROM).

1.5 Explain the differences between a local-area network (LAN) and a wide-area network (WAN). How do they work with each other?

1.6 What is the total number of communication lines needed for a fully connected point-to-point network of eight computers? Nine computers? Ten computers? What is a general formula for determining this result?

1.7 Give examples of the two types of Java comments and explain the differences between them.

1.8 Why are the following valid Java identifiers not considered good identifiers?

a. `q`

b. `totVal`

c. `theNextValueInTheList`

1.9 Identify each of the following situations as a compile-time error, runtime error, or logical error.

a. multiplying two numbers when you meant to add them

b. dividing by zero

c. forgetting a semicolon at the end of a programming statement

d. spelling a word wrong in the output

e. producing inaccurate results

f. typing a `{` when you should have typed a `(`

1.10 How many bits are needed to store a color picture that is 400 pixels wide and 250 pixels high? Assume color is represented using the RGB technique described in this chapter.

programming projects

1.1 Enter, compile, and run the following application:

```
public class Test
{
   public static void main (String[] args)
   {
      System.out.println ("An Emergency Broadcast");
   }
}
```

1.2 Introduce the following errors, one at a time, to the program from the Programming Project 1.1. Record any error messages that the compiler produces. Fix the previous error each time before you introduce a new one. If no error messages are produced, explain why. Try to predict what will happen before you make each change.

a. change `Test` to `test`

b. change `Emergency` to `emergency`

c. remove the first quotation mark in the string

d. remove the last quotation mark in the string

e. change `main` to `man`

f. change `println` to `bogus`

g. remove the semicolon at the end of the `println` statement

h. remove the last brace in the program

1.3 Write an application that prints, on separate lines, your name, your birthday, your hobbies, your favorite book, and your favorite movie. Label each piece of information in the output.

1.4 Write an application that prints the phrase `Knowledge is power`:

a. on one line

b. on three lines, one word per line, with the words centered relative to each other

c. inside a box made up of the characters = and |

1.5 Write an application that prints the following diamond shape. Don't print any unneeded characters. (That is, don't make any character string longer than it has to be.)

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

1.6 Write an application that displays your initials in large block letters. Make each large letter out of the corresponding regular character. For example:

```
JJJJJJJJJJJJJJ   AAAAAAAAA    LLLL
JJJJJJJJJJJJJJ  AAAAAAAAAAA   LLLL
        JJJJ     AAA     AAA   LLLL
        JJJJ     AAA     AAA   LLLL
        JJJJ     AAAAAAAAAAA   LLLL
J       JJJJ     AAAAAAAAAAA   LLLL
JJ      JJJJ     AAA     AAA   LLLL
 JJJJJJJJJJ      AAA     AAA   LLLLLLLLLLLLLL
  JJJJJJJJ       AAA     AAA   LLLLLLLLLLLLLL
```

answers to self-review questions

1.1 The hardware of a computer system is its physical parts such as a circuit board, monitor, or keyboard. Computer software are the programs that are executed by the hardware and the data that those programs use. In order to be useful, hardware requires software and software requires hardware.

1.2 The operating system provides a user interface and coordinates the use of resources such as main memory and the CPU.

1.3 The information is broken into pieces, and those pieces are represented as numbers.

1.4 In general, N bits can represent 2^N unique items. Therefore:

a. 2 bits can represent 4 items because $2^2 = 4$.

b. 4 bits can represent 16 items because $2^4 = 16$.

c. 5 bits can represent 32 items because $2^5 = 32$.

d. 7 bits can represent 128 items because $2^7 = 128$.

1.5 There are eight bits in a byte. Therefore:

a. 8 bytes = 8 * 8 bits = 64 bits

b. 2 KB = 2 * 1,024 bytes = 2,048 bytes = 2,048 * 8 bits = 16,384 bits

c. 4 MB = 4 * 1,048,576 bytes = 4,194,304 bytes = 4,194,304 * 8 bits = 33,554,432 bits

1.6 The two main hardware components are main memory and the CPU. Main memory holds the currently active programs and data. The CPU retrieves individual program instructions from main memory, one at a time, and executes them.

1.7 A memory address is a number that identifies a particular memory location in which a value is stored.

1.8 Main memory is volatile, which means the information that is stored in it will be lost if the power supply to the computer is turned off. Secondary memory devices are nonvolatile; therefore the information that is stored on them is retained even if the power goes off.

1.9 A file server is a network computer that is dedicated to storing and providing programs and data that are needed by many network users.

1.10 Counting the number of unique connections in Figure 1.16, there are 10 communication lines needed to fully connect a point-to-point network of five computers. Adding a sixth computer to the network will require that it be connected to the original five, bringing the total to 15 communication lines.

1.11 The word *Internet* comes from the word *internetworking*, a concept related to wide-area networks (WANs). An internetwork connects one network to another. The Internet is a WAN.

1.12 Breaking down the parts of each URL:

a. `chs` is the name of the computer within the `mcps.org` domain, which represents Montgomery County public schools in Virginia. The `org` top-level domain indicates that it is an organization. This URL is requesting a file called `math.htm` from within a subdirectory called `Faculty`.

b. `java` is the name of a computer (Web server) at the `sun.com` domain, which represents Sun Microsystems, Inc. The `com` top-level domain indicates that it is a commercial business. This URL is requesting a file called `index.html` from within a subdirectory called `products`.

1.13 High-level languages let a programmer write program instructions in English-like terms that are relatively easy to read and use. However, in order to execute, a program must be translated into machine language, which is a series of bits that are basically unreadable by humans. A high-level language program must be translated into machine language before it can be run.

1.14 Java bytecode is low-level Java source code. The Java compiler translates the source code into bytecode, which can then be executed using the Java interpreter. The bytecode might travel across the Web before being executed by a Java interpreter that is part of a Web browser.

1.15 White space is the spaces, tabs, and newline characters that separate words and symbols in a program. The compiler ignores extra white space, so it doesn't affect execution. However, white space can make a program readable to humans.

1.16 All of the identifiers shown are valid except `12345` (since an identifier cannot begin with a number) and `black&white` (since an identifier cannot contain the character `&`). The identifiers `RESULT` and `result` are both valid, but should not be used together in a program because they differ only by case. The underscore character (as in `answer_7`) is a valid part of an identifier.

1.17 Syntax rules define how the symbols and words of a programming language can be put together. Semantics determine what will happen when that instruction is executed.

1.18 A black-and-white picture can be drawn using a series of dots, called pixels. Pixels with a value of 0 are displayed in white and pixels with a value of 1 are displayed in black. A realistic black-and-white photo can be produced on a computer screen using thousands of pixels.